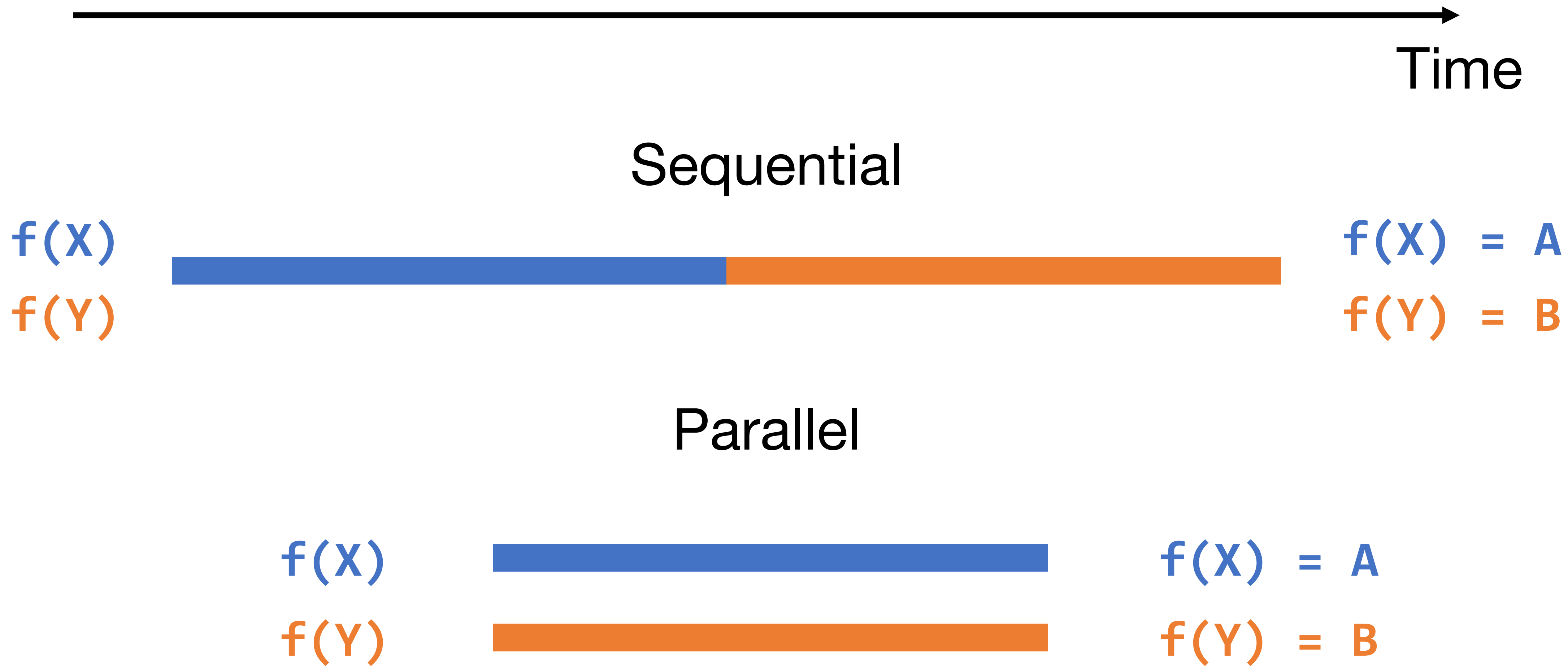
Concurrency

CS 240

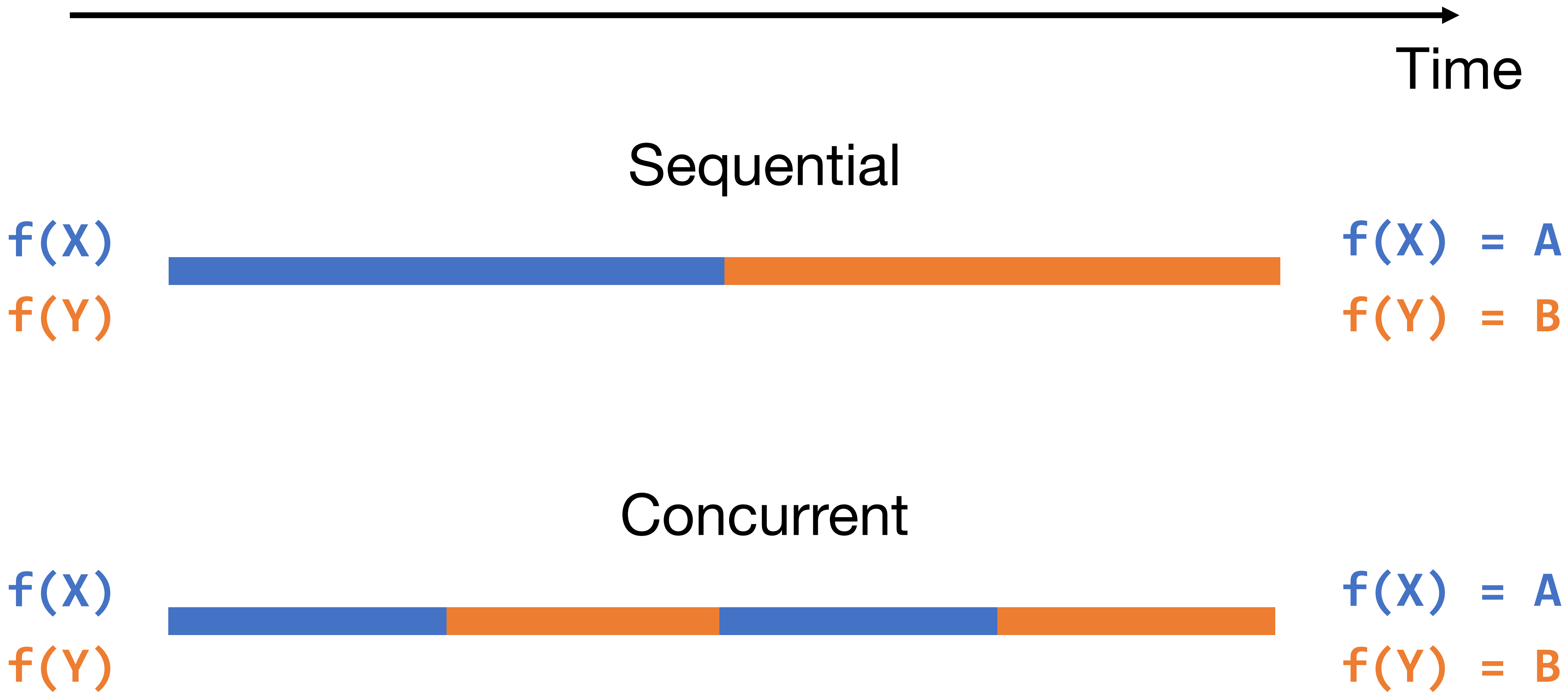
What is Concurrency?

It’s like parallel that’s not in parallel

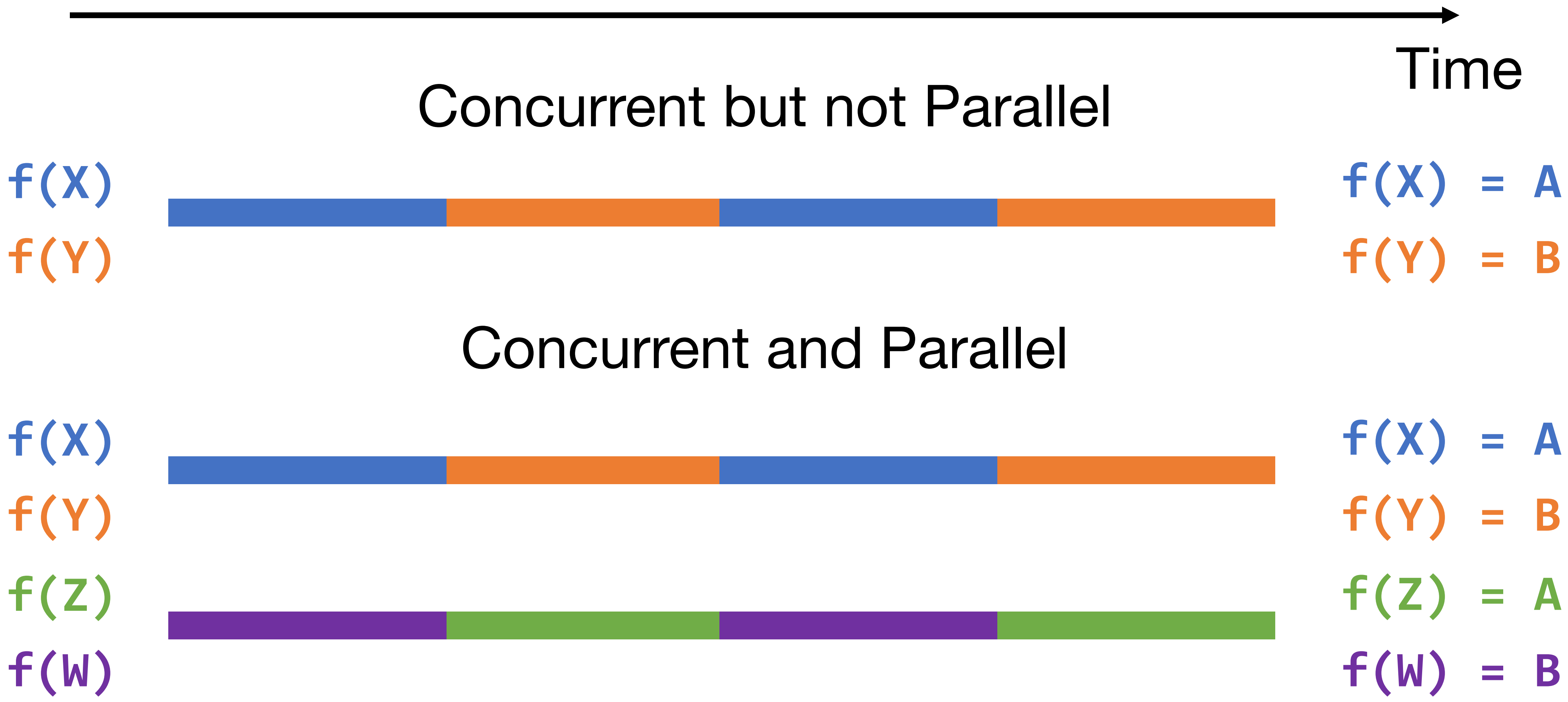
What is Parallelism?



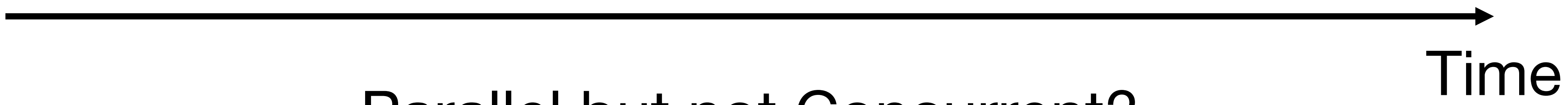
What is Concurrency?



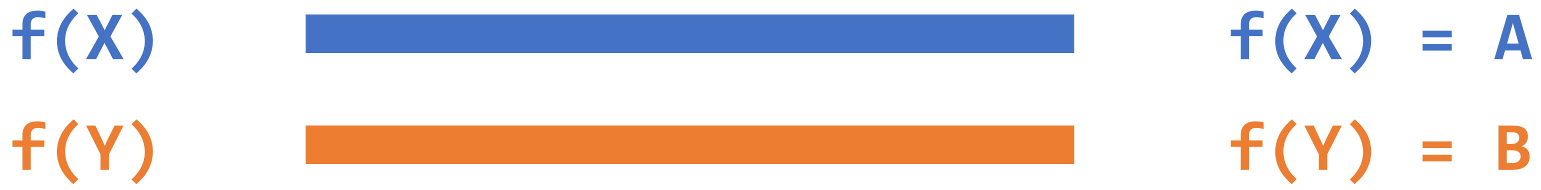
Concurrency Could be Parallel but not Always



Parallel is Always Concurrent



Parallel but not Concurrent?

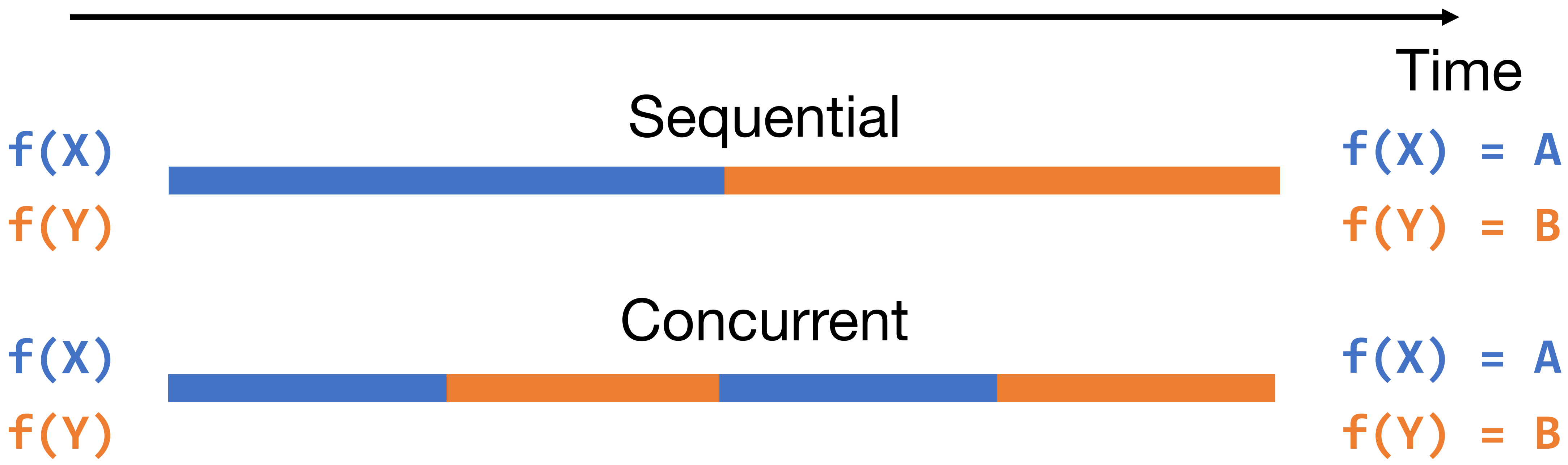


Nope … still concurrent

$$\text{Parallel} \rightarrow \text{Concurrent}$$
$$\text{Concurrent} \nrightarrow \text{Parallel}$$

Why Care about Concurrency

If something concurrent but not parallel takes as much time as something sequential, why make it concurrent?



Concurrency is a *Design* Pattern

“Concurrency is about dealing with lots of things at once.
Parallelism is about doing lots of things at once.”

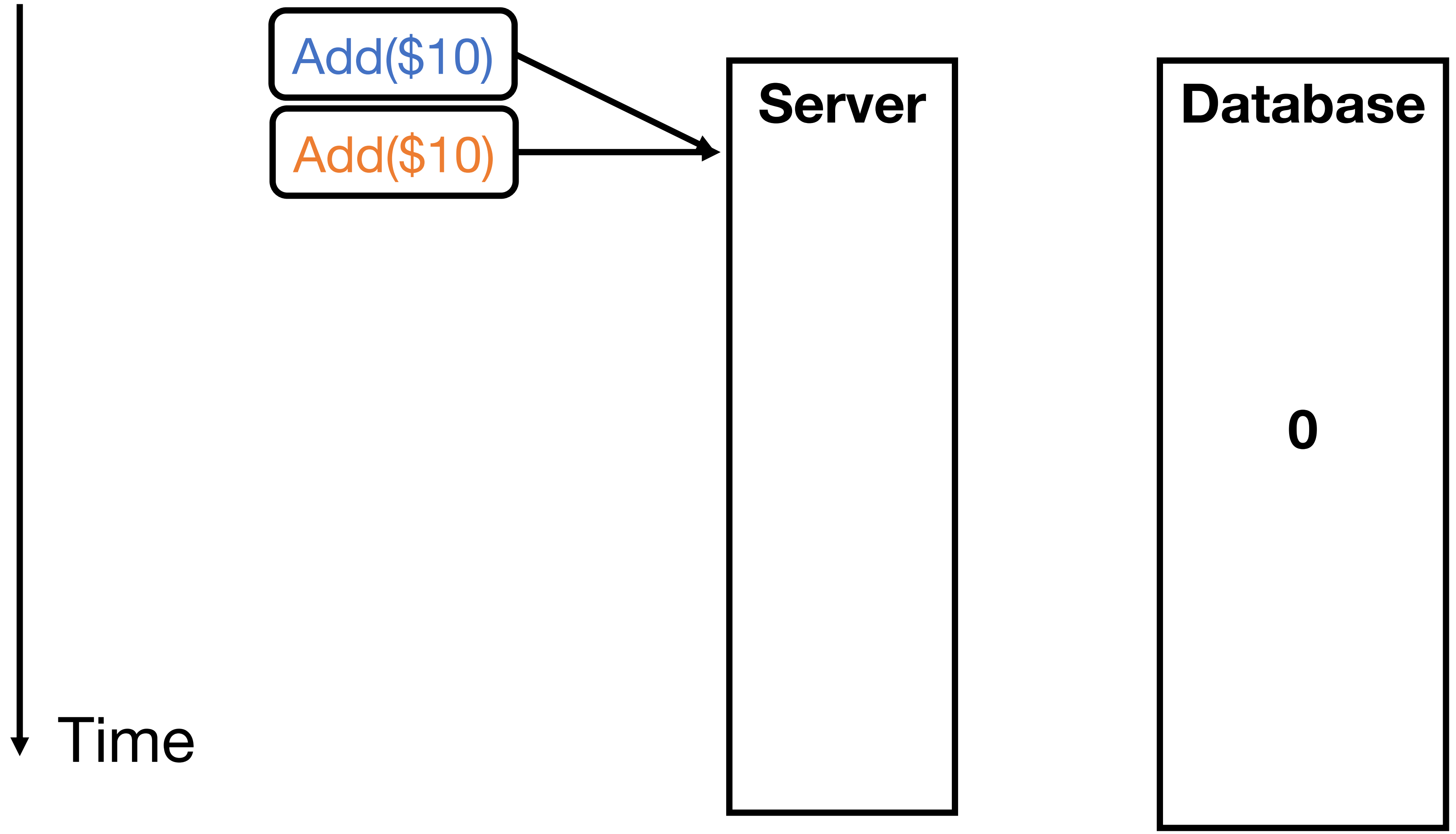
- Rob Pike

Distributed Systems are Unpredictable

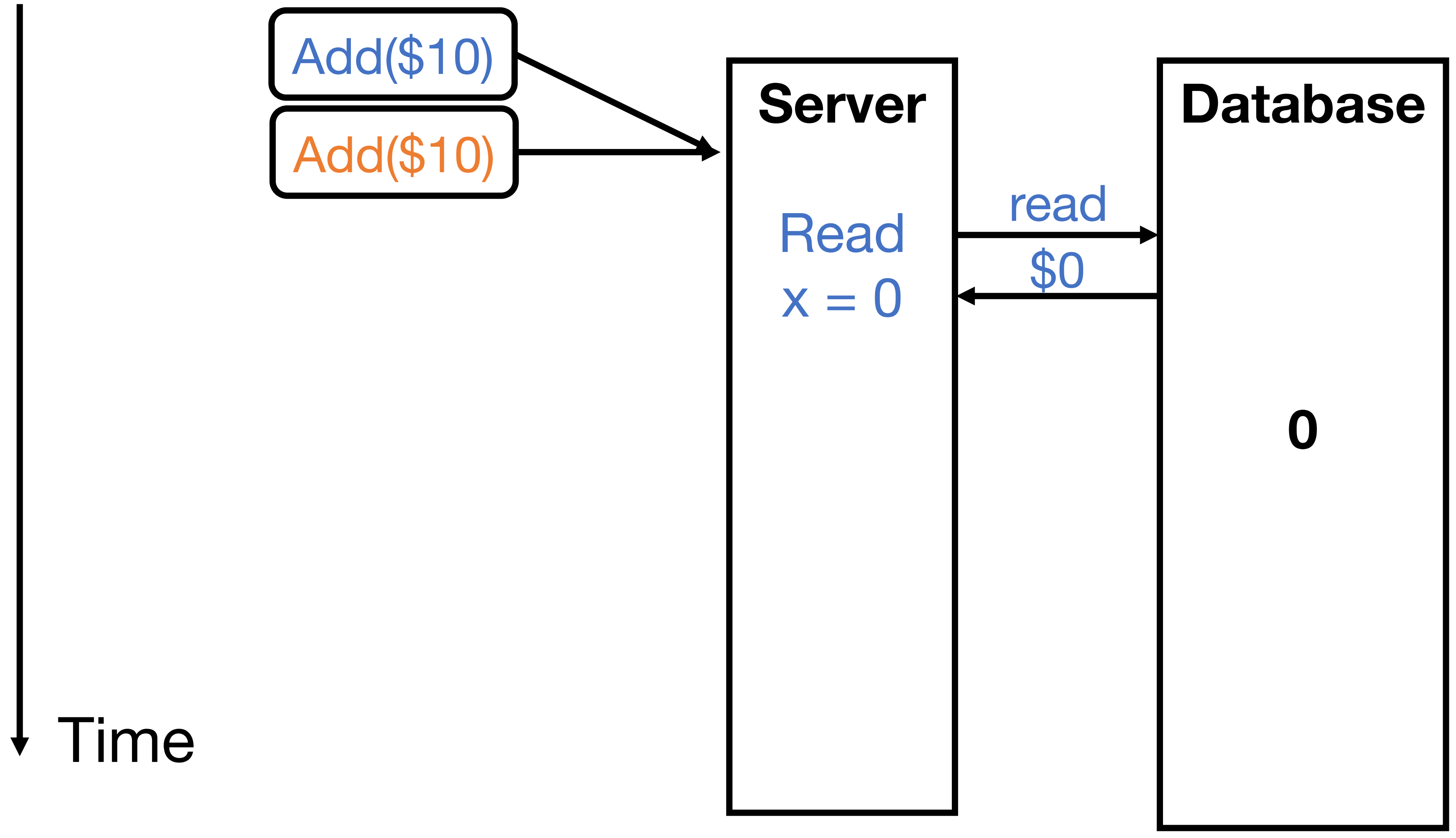
Servers need to react to:

- Others servers
- Crashes
- Users
- …

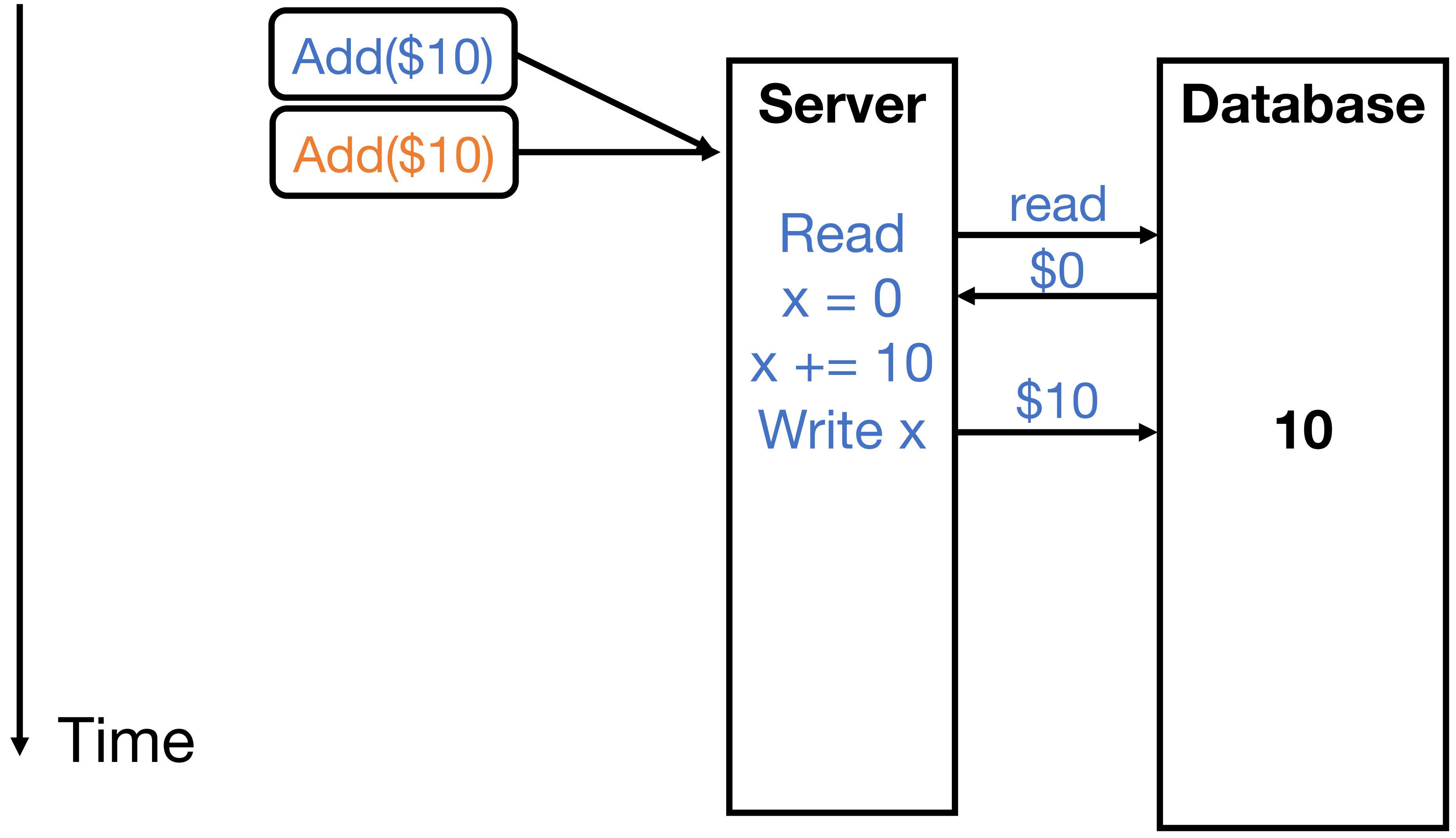
Making Bank Deposits Concurrent (1/5)



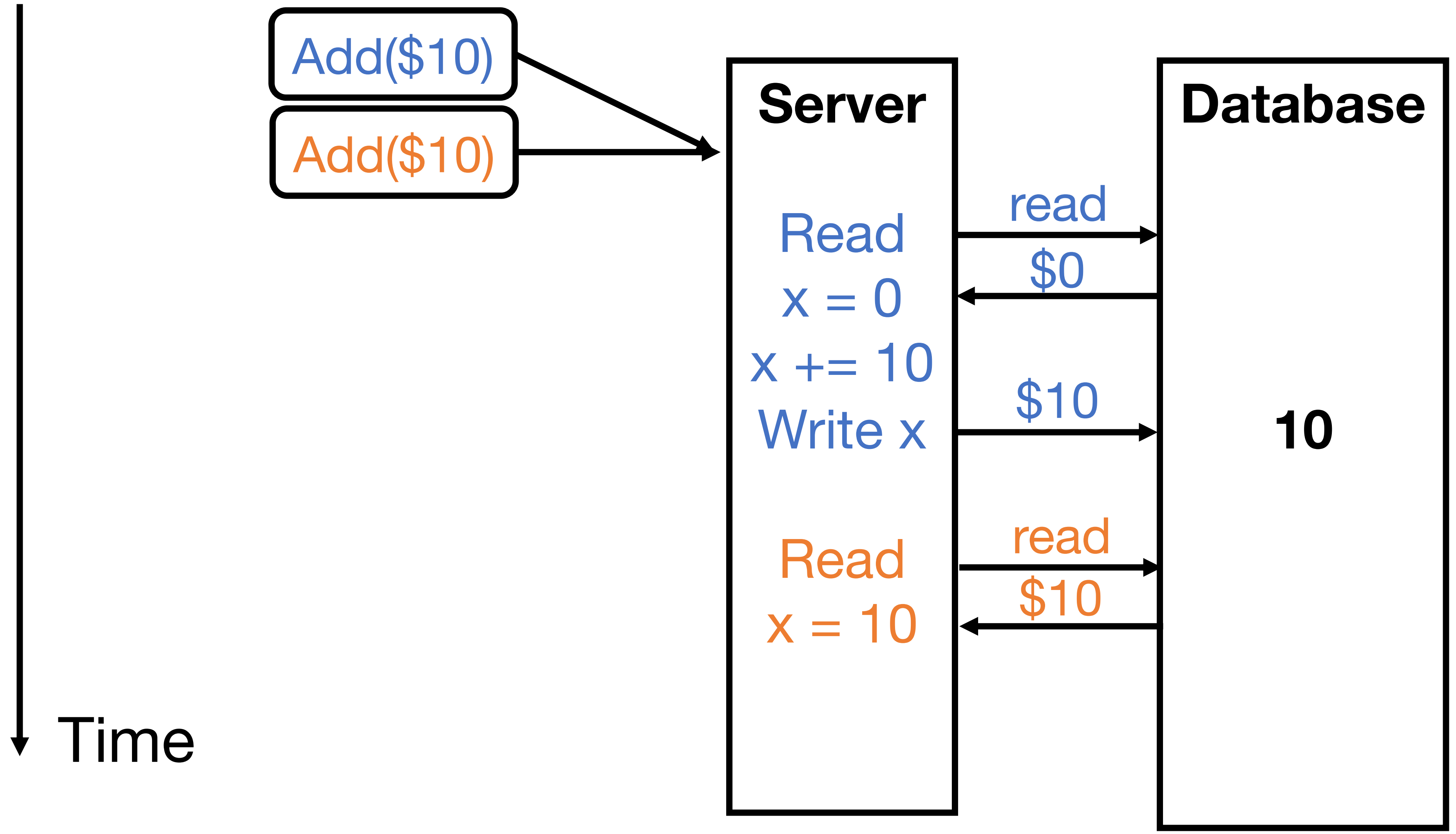
Making Bank Deposits Concurrent (2/5)



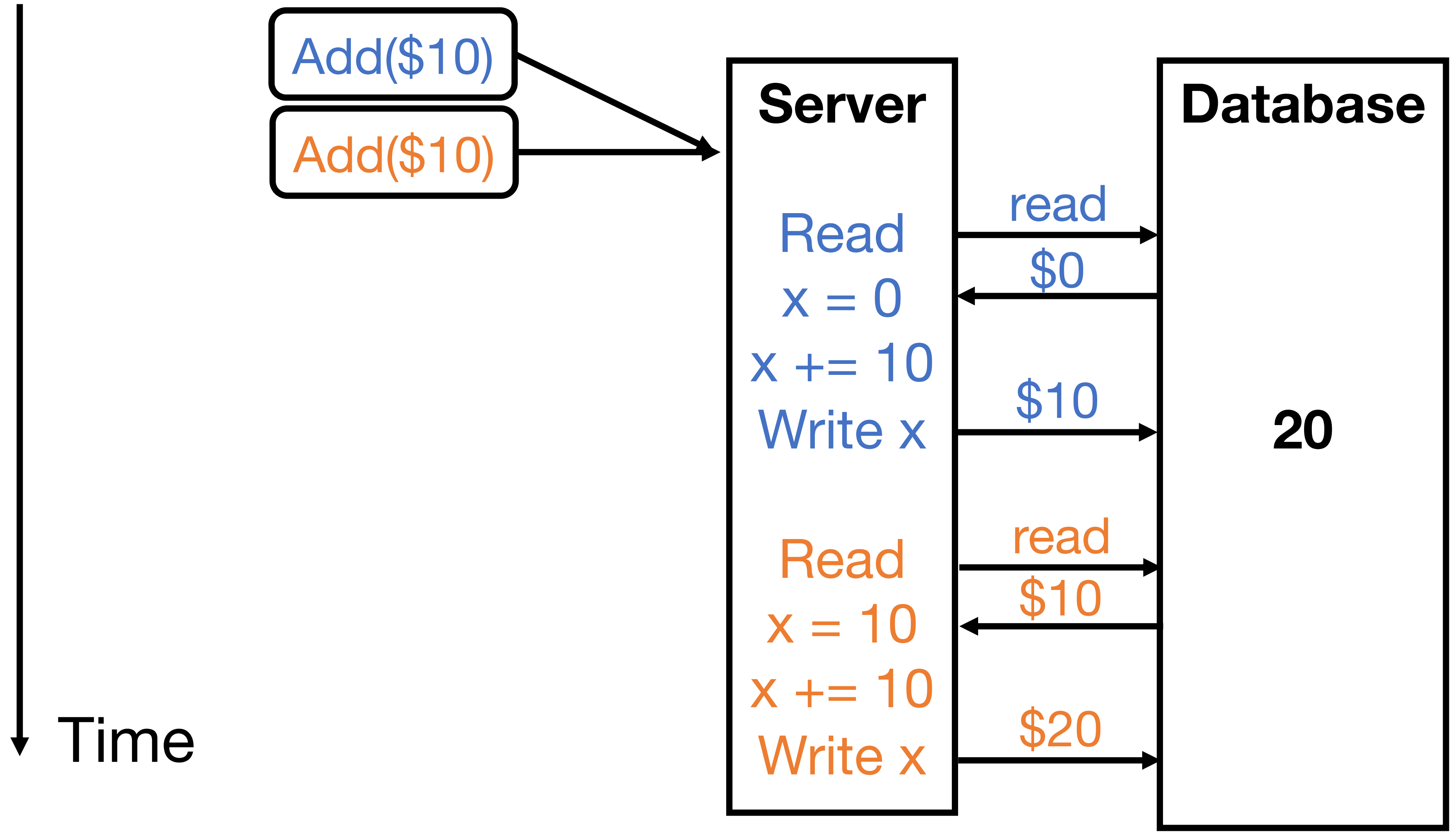
Making Bank Deposits Concurrent (3/5)



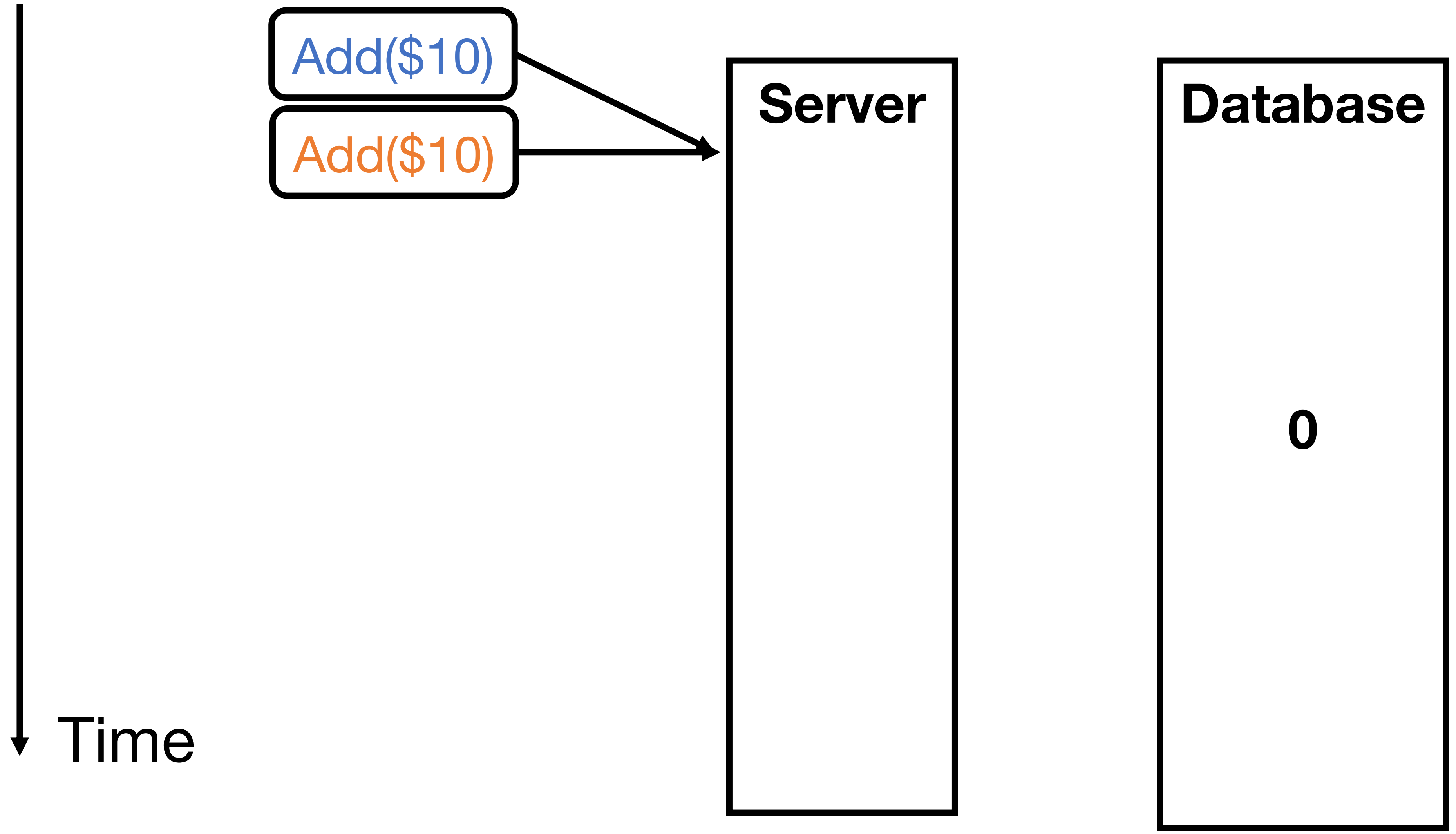
Making Bank Deposits Concurrent (4/5)



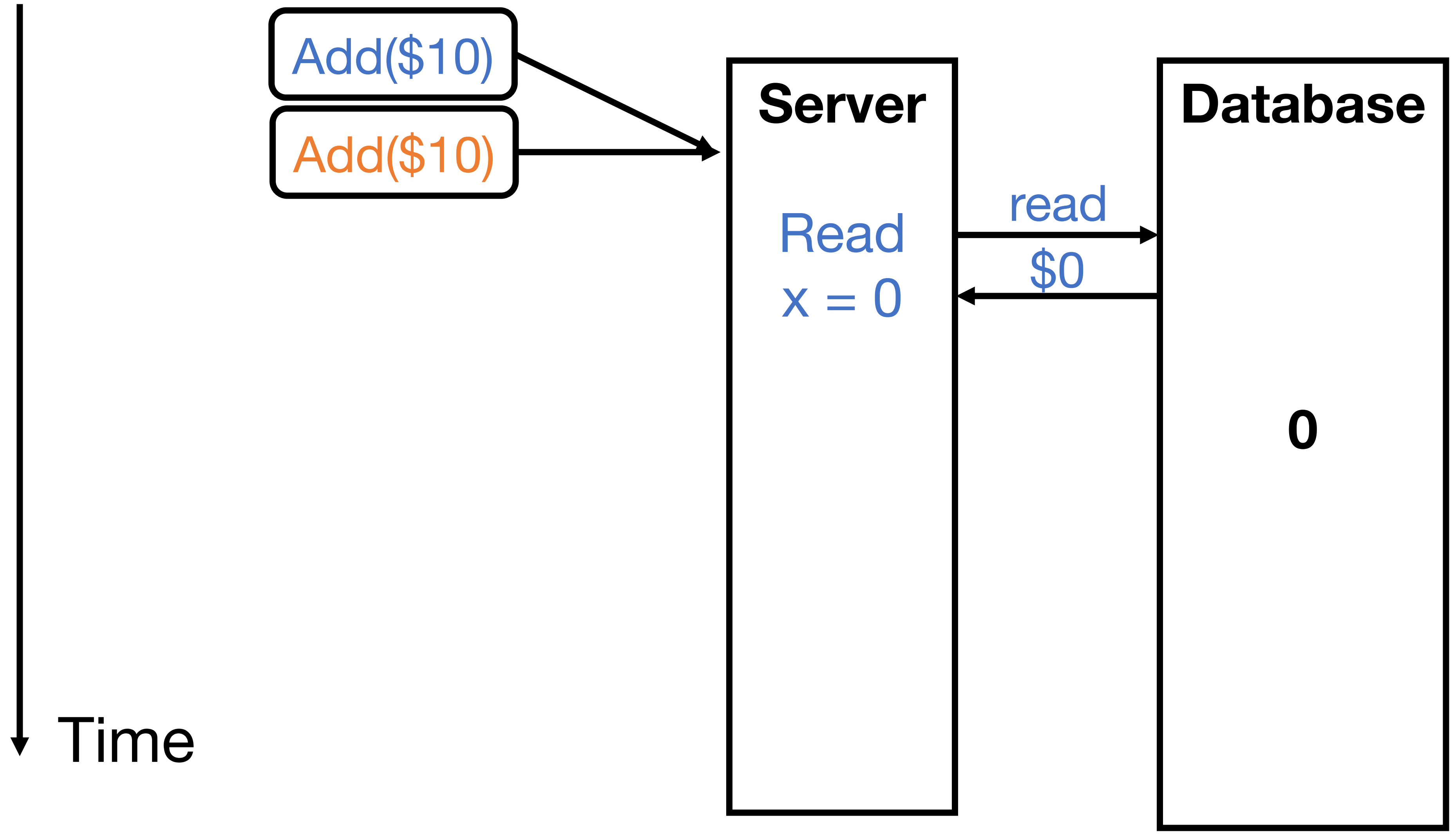
Making Bank Deposits Concurrent (5/5)



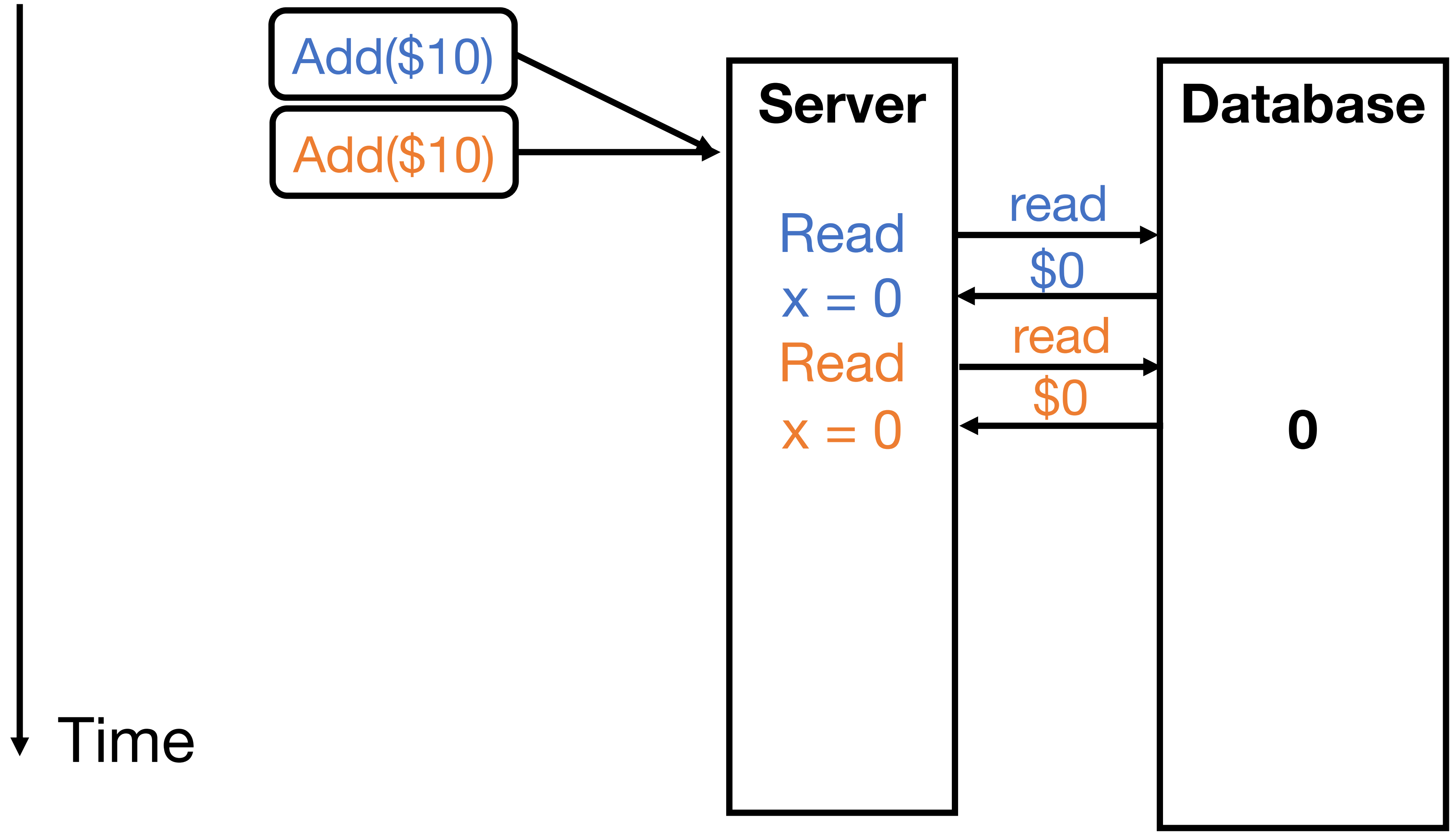
Concurrent Bank Deposits! Yay? (1/5)



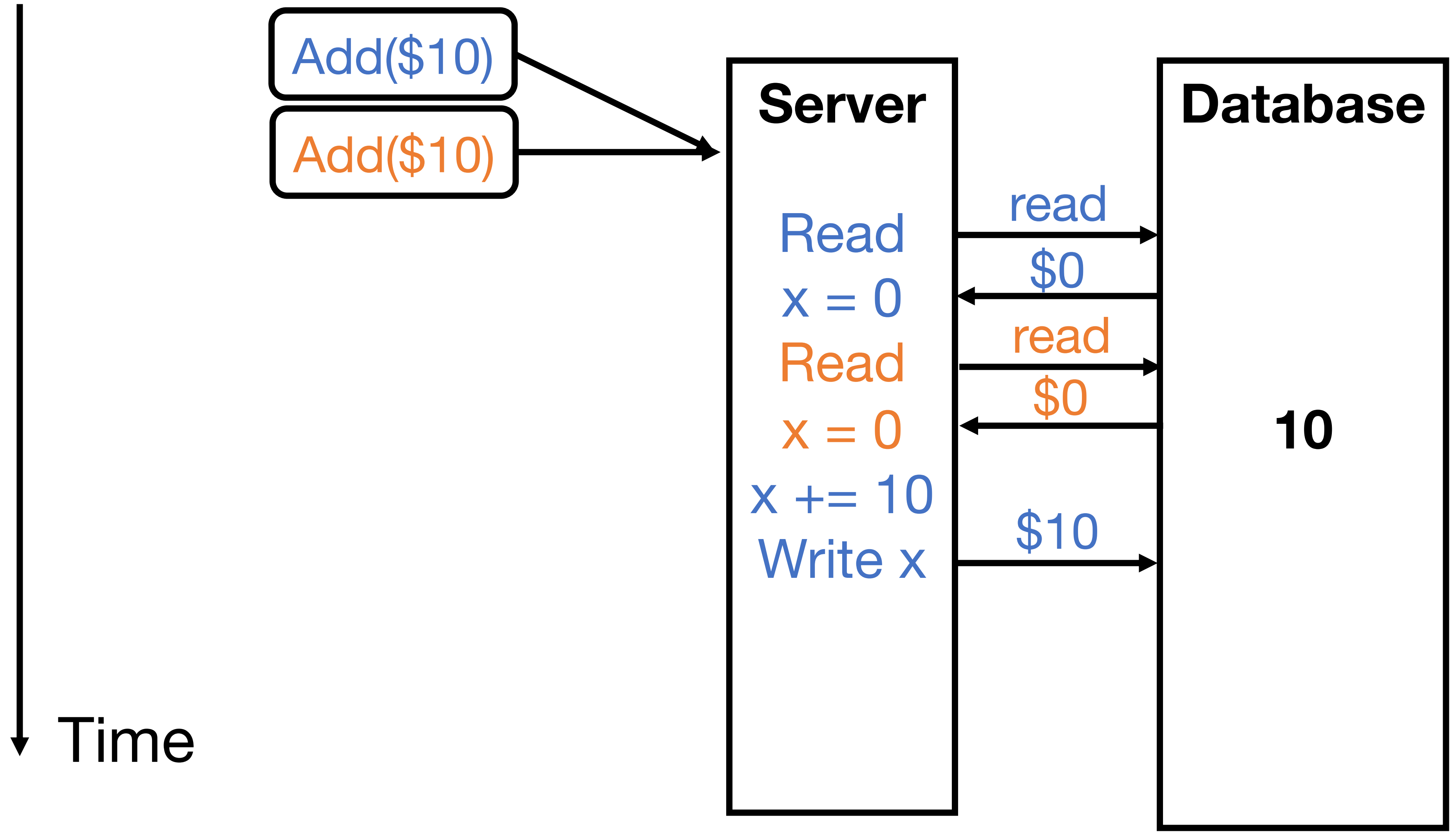
Concurrent Bank Deposits! Yay? (2/5)



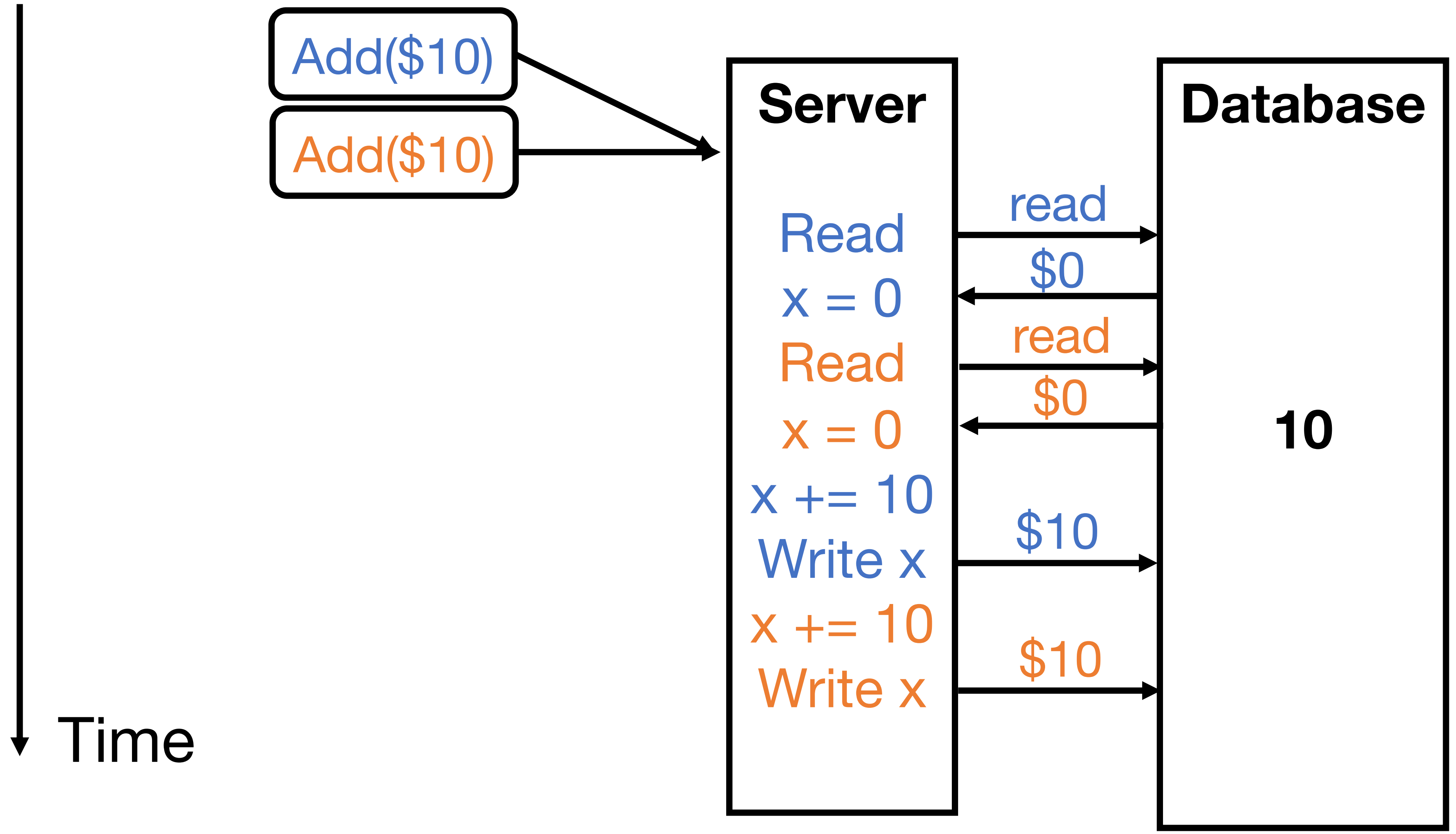
Concurrent Bank Deposits! Yay? (3/5)



Concurrent Bank Deposits! Yay? (4/5)



Concurrent Bank Deposits! Yay? (5/5)



Concurrency Needs to be Synchronized

Locks – limit access using shared memory
Channels – pass information using a queue

Visualize Everything We’ve Learned

And also see many different methods of achieving synchronization:
http://divan.github.io/posts/go_concurrency_visualize/

RPCs in Go

Networked battleship game

CS 240

What is a RPC (Remote Procedure Call)?

RPC means a client will execute some function on a remote server

- Client make a local requests with some parameters
- RPC library encodes the request and parameters, send them to server
- Server decodes the request and parameters
- Procedure is executed on the server
- Server sends back the reply to the client

RPC exercise

We will use the net/rpc package to implement a client and server

https://golang.org/pkg/net/rpc/

Server side:

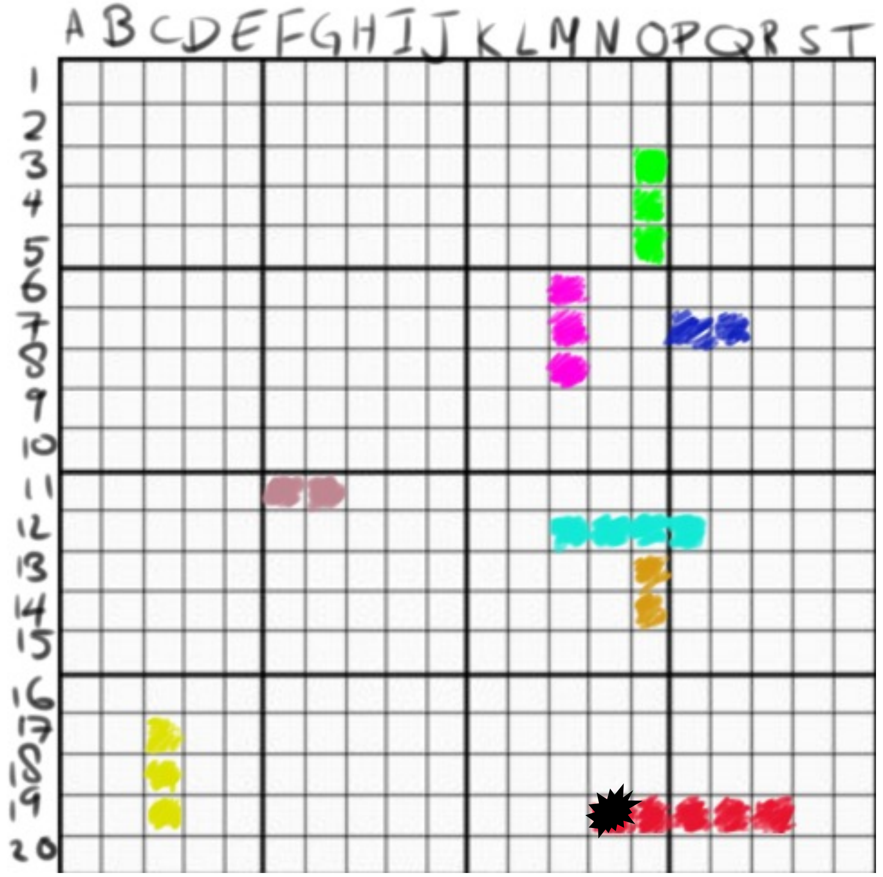
- Create the server instance
- Define the procedure
- Listen for incoming requests

Client side:

- Create the client instance and connect to the server
- Make the RPC

Battleship

- A grid map on which you place your ships



"Deploy attack on (N, 19)!"

Goal: Find and sink all enemy ships

Today’s task: Implement a Battleship client

- Project files available on the Campuswire
- We will run a central server

- Implement the client (client.go) and test it against other students

Task 1 and 2

- Establish connection to the server
  - See https://golang.org/pkg/net/rpc/ example “rpc.DialHTTP”
  - Must return a rpc.Client object

- Make the JoinGame request
  - You want to call the remote BattleshipsService.JoinGame function
  - Parameters PublicPlayer and JoinGameRequest are defined in common.go
  - See https://golang.org/pkg/net/rpc/ example “client.Call”

Task 3

- Implement the attack server
    - Tasks 1 and 2 were making requests as a client, now must accept requests
    - See https://golang.org/pkg/net/rpc/
        - Examples “rpc.Register” and “rpc.HandleHTTP”
    - Create a listener to serve requests on a separate goroutine

Task 4

- Implement the turn logic
  - Hint: The turn logic can be achieved with Channels, Locks or WaitGroups
  - Hint 2: When the other player attacks, you get a “token” to make one attack

After implementation is complete, you can run against other players